RESEARCH PLAN PROPOSAL

Developing an Efficient Query Processing Technique in Cloud Based MapReduce System

For Registration to Doctor of Philosophy

IN THE FACULTY OF COMPUTER SCIENCE

THE IIS UNIVERSITY, JAIPUR

Submitted by

VANDANA VIJAY
ICG/2017/24570

Under the Supervision of

Dr. Ruchi Nanda
Associate Professor (CS & IT)

Department of Computer Science and Information Technology

October, 2018

Developing an Efficient Query Processing Technique in Cloud Based MapReduce System

1. Introduction

In the last few years, the global data volume has been increasing at a very high rate due to the proliferation of technologies including Cloud Computing and Internet of Things. This leads to the accumulation of huge amount of data in today's environment on the regular basis. MapReduce has becoming a popular computing model for processing and generating huge data sets on a cluster of machines. Cloud computing provides a reliable, available and scalable environment for the processing of this huge data. The traditional techniques utilized for processing of queries are not suitable for big data scenario as they cannot handle large amount of data. The performance of queries is a major concern. So in current circumstances there has been a strong focus on the development of techniques such as MapReduce which can handle enormous amount of data in an efficient manner.

Query Processing in Traditional Database System

The management of huge data is an important concern in any organization today. Database Management System (DBMS) plays a crucial role in maintaining this data. It consists of a collection of interrelated data and a collection of programs to access that data. Traditional DBMS stores data in the form of rows and columns similar to a table, which is referred to as Relational Database Management System (RDBMS). The function of a database system is, to respond efficiently to user queries for the desired information. In DBMS, query processor module is responsible for executing database queries and obtains desired information from a database system in a reliable and efficient way. Query processing is the procedure of transforming a high-level language query such as SQL (Structured Query Language) into a correct and efficient execution plan expressed in low-level language. A query optimizer selects the most appropriate plan that is used in responding to a database request.

Query processing is a three-step process that transforms a high-level language(SQL) query into an equivalent and more efficient lower-level query of relational algebra (Silberschatz et al. 2011). It is shown in Figure 1. In the first phase, the syntax analyzer reads the SQL query and checks for correctness. This phase is called syntax checking phase. It uses the grammar of SQL as input and the parser portion of the query processor check the syntax and verify whether the relation and attributes of the requested query are defined in the database or not. Then, the SQL query is translated into an algebraic expression using various rules. In the second phase called

optimization phase, the relational algebraic expression are passed to the query optimizer. At this stage, the cost model and several other estimation formulas are used to rewrite the query. The query optimizer then generates an action plan also called an execution plan. In the third phase called evaluation phase, the query-execution engine takes an (optimal) evaluation plan, executes that plan, and returns the answers to the query.

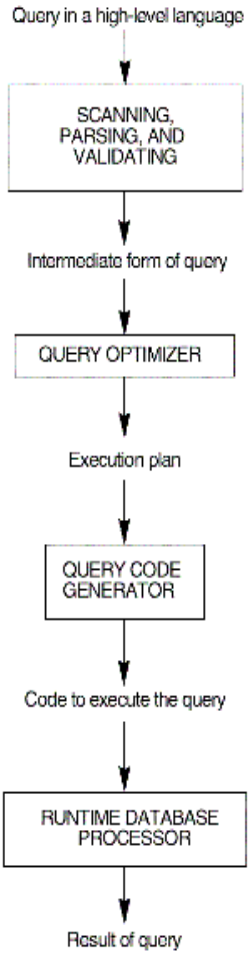


Figure 1: Steps in Query Processing (Elmasri and Navathe, 2016)

In the last few years, the global data volume has been increasing at a very high rate. This data includes not only structured data but also contain unstructured data such as text, images, audio, videos and emails. RDBMS cannot handle such huge volume of data because they are designed for steady data retention, have inflexible schemas and can handle only gigabytes of data (Bachhav et al. (2017), Nanda (2015), Doulkeridis et al. (2014), Bobrowski et al. (2011). They have grown overly complex, difficult to manage, and are struggling today. Hence, in order to handle such huge volume of heterogeneous data, companies are now coming up with alternatives to RDBMS. One of the

widely accepted solutions is MapReduce, which has attracted great attention due to its flexibility, parallel processing, ease of programming, and fault tolerance features.

MapReduce

MapReduce is a programming model suitable for processing of huge data. It decomposes large manipulation jobs into independent tasks that can be executed in parallel, across a cluster of servers. The results of tasks can be joined together to compute the final results.

The MapReduce contains two tasks, namely, Map and Reduce which are described below:

1. Map: In this task, a block of data is read and processed to produce key-value pairs as intermediate outputs. The outputs of a map job are in the form of key-value pairs, which are then fed to the Reducer.
2. Reduce: Here, reducer receives the key-value pairs from multiple map jobs. It aggregates those intermediate key-value pairs into a smaller set of key-value pairs to produce the final output.

The features of MapReduce (Yang et al. (2007), Blanas et al. (2010)) are as follows:

1. Parallel processing: MapReduce divides the job among multiple nodes and each node participates in the processing of a portion of the job simultaneously.
2. Fault-tolerant: The distributed file system replicates every data block multiple times on different nodes. Hence, input file can easily survive the node failure.
3. Load balancing: Each node is given a certain number of initial tasks only. Every time a node finishes a task, it demands more tasks. In this way, faster nodes get more tasks than slower ones.
4. Schema Flexibility: MapReduce support schema flexibility, by allowing the processing of structured and unstructured data easily.
5. Scalability: MapReduce concern less about the per-node efficiency but focus more on scalability. As long as the framework scales, any moderate performance loss on a single node can be compensated by simply employing more hardware.
6. Set-Oriented Keys and Values: MapReduce can process thousands of file chunks in parallel. Developers can see the data only as keys and values, no longer raw bits and bytes, nor file descriptors.

Query Processing in MapReduce-Based Systems

There are three important phases of MapReduce shown in figure 2 and are described below:

1. Mapping: In this phase, input data is divided into fixed-size splits. Each split is then passed to a mapping function which in turn, produces output values.

2. Shuffling: This phase consumes output of mapping phase. It consolidates the relevant records from the output of Mapping phase.
3. Reducing: Output values from shuffling phase are aggregated in this phase to return a single output value.

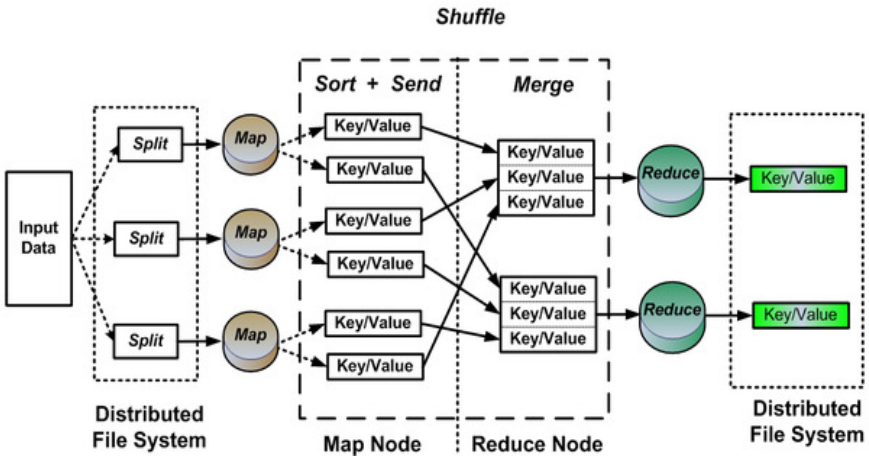


Figure 2: Representation of MapReduce Architecture [Akhtar et al. 2017]

The architecture of MapReduce framework is described below.

The Map function takes one pair of data with a type in one data domain, and returns a list of pairs in a different domain:

$Map(k1,v1) \rightarrow list(k2,v2)$

The Map function is applied in parallel to every pair (keyed by k1) in the input dataset. This produces a list of pairs (keyed by k2) for each call. Then, MapReduce framework collects all pairs with same key (k2) from all lists and groups them together, creating one group for each key.

The Reduce function is then applied in parallel to each group, which in turn produces a collection of values in the same domain:

$Reduce(k2, list\ (v2)) \rightarrow list(v3)$

Each Reduce call typically produces either one value v3 or an empty return, though one call is allowed to return more than one value. The returns of all calls are collected as the desired result list. Thus the MapReduce framework transforms a list of (key, value) pairs into a list of values.

This architecture can be explained by taking an example of ‘occurrence’ that counts the appearance of each word in a set of documents. Figure 3 shows the pseudocode for word count occurrence.

```
map (String key, String value):
        // key: document name
        // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");
```

```
reduce (String key, Iterator values):
        // key: a word
        // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
        Emit (key, AsString(result));
```

Figure 3: Pseudo Code for word count occurrence

Each document splits into words and each word is counted by the map function, using the word as the result key. The framework puts together all the pairs with the same key and feeds them to the same call to reduce. Hence, reduce function calculates sum of all its input values to find the total appearances of that word.

The open source implementation of MapReduce programming model is Apache Hadoop. It is mainly used for distributed storage and processing of dataset of bigdata using the MapReduce programming model. The core of Apache Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), and a processing part, known as MapReduce. Hadoop splits files into large blocks and distributes them across nodes in a cluster or group. It then transfers packaged code into nodes to process the data in parallel.

MapReduce over Cloud

In today's environment almost all the companies have migrated their applications and data to the cloud due to the popularity of the Internet along with a sharp increase in the network bandwidth. This has resulted in the generation of huge amounts of data on regular basis. MapReduce has becoming a popular computing model for cloud computing platform and is helpful to examine this large amount of data. Since the current design of MapReduce does not consider virtualization so Cloud computing provides the virtualized environment to it where multiple virtual machines (VMs) share resources such as disks, networks, and main memory. As per Hwang et al. (2018) Cloud computing is a cost-effective and elastic computing environment, enables the automatic provisioning of computing resources on demand as the workload dynamically changes. It is defined as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction (NIST, 2015). Hence, running MapReduce on cloud computing is becoming popular as it provides a reliable, available and scalable environment for the processing of this huge data.

The essentials characteristics of Cloud Computing are shown in figure 4 and are described below:

1. On Demand Self Service: Cloud Computing allows the users to use web services and resources on demand.
2. Broad Network Access: Since cloud computing is completely web based, it can be accessed from anywhere and at any time.
3. Rapid Elasticity: It is very easy to scale the resources vertically or horizontally at any time. Scaling of resources means the ability of resources to deal with increasing or decreasing demand. The resources being used by customers at any given point of time are automatically monitored.
4. Resource Pooling: Cloud computing allows multiple tenants to share a pool of resources. One can share single physical instance of hardware, database and basic infrastructure.
5. Measured Service: Resource usage can be monitored, controlled and reported, providing transparency for the provider and consumer.

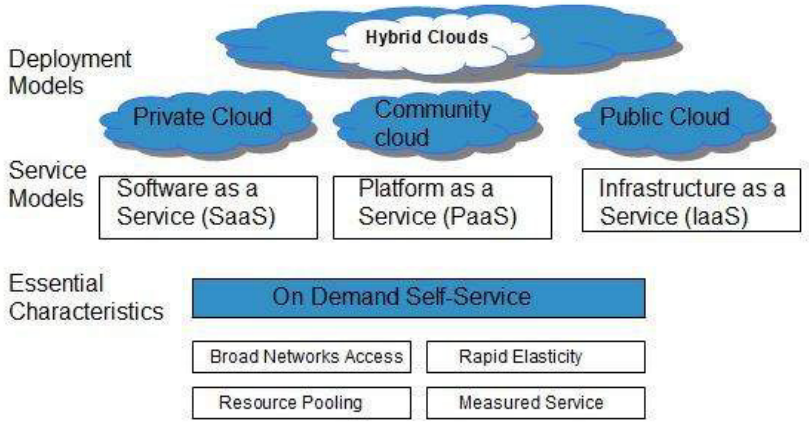


Figure 4: Characteristics of cloud computing (NIST, 2015)

The three basic service models (Almorsy et al. (2016)) are as follows:

1. Infrastructure-as-a-Service (IaaS): Here, cloud providers deliver computation resources, storage and network as a internet-based services. This service model is based on the virtualization technology. Amazon EC2 is the most familiar IaaS provider.
2. Platform-as-a-Service (PaaS): Here, cloud providers deliver platforms, tools and other business services that enable customers to develop, deploy and manage their own applications. These services do not require platforms or support tools on their local machines. The PaaS model may be hosted on top of IaaS model or on top of the cloud infrastructures directly. Google Apps and Microsoft Windows Azure are the most known PaaS.
3. Software-as-a-Service (SaaS): Here, cloud providers deliver applications hosted on the cloud infrastructure as internet-based service for end users, without requiring installing the applications on the customers' computers. This model may be hosted on top of PaaS, IaaS or directly hosted

on cloud infrastructure. SalesForce and CRM (Customer Relationship Management) are examples of the SaaS provider.

Enterprises can choose to deploy applications on three types of cloud described as below:

1. Public cloud: A cloud in which service provider offer their resources as services to the general public. There is no need to invest capital on infrastructure and no risks involved in shifting to infrastructure.
2. Private clouds: They are designed for exclusive use by a single organization. A private cloud may be built and managed by the organization or by external providers. It offers the highest degree of control over performance, reliability and security.
3. Hybrid clouds: A hybrid cloud is a combination of public and private cloud models that tries to address the limitations of each approach. In a hybrid cloud, part of the service infrastructure runs in private cloud while the remaining part runs in public cloud. Hybrid clouds offer more flexibility than both public and private clouds.

There are various factors that decide the performance of query such as joining, partitioning, pre-processing, caching, indexing, colocation, pipelining, materialization, etc. Among them we are focusing on four important factors which are most common, frequently used and costly in terms of I/O and CPU usage (Blanas et al. (2010), Doulkeridis et al. (2014), Nanda et al. (2017), Bachhav et al. (2017)). These are described as follows:

1. Join operation: It is one of the fundamental relational database query operations. It facilitates the retrieval of information from two different relations based on a Cartesian product of the two relations. Three most commonly used categories of joins are: Theta join, Equijoin and Natural Joins. The Join operation in MapReduce can be group into two categories:
   - Map-side join: Here, the join operation is performed in their map side phase
   - Reduce-side join: The join operation is performed in their reduce side phase.
2. Caching: Cache refers to the intermediate data which is produced by the worker nodes/processes during the execution of any MapReduce task. A part of cached data is stored in a Distributed File System (DFS).
3. Partitioning: It is the process of determining which reducer instance will receive which intermediate keys and values. Each mapper must determine for all of its output (key, value) pairs which reducer will receive them.

4. Pipelining: When a query is composed of several relational algebra operators, the result of one operator is sometimes pipelined to another operator without creating a temporary relation to hold the intermediate result. This process is known as pipelining.

2. Review of Literature

Papers related to MapReduce framework have been studied which are helpful in understanding the techniques for improving the performance of MapReduce in today environment.

Join Processing:

Liu and Li (2018) proposed two methods to effectively process binary theta-join and multi-way theta-join in distributed environment. The first method is a filter strategy called Max and Min value-based filter method (MMF), which can filter much irrelevant records from input data sets before computing cross product of theta-join, then reduce the computation cost. The second method is Divide and Merge (DM) method which can be used in both binary theta-join and multi-way theta-join. This method uses the join attribute to do cross product and uses equi-join to combine the result with other output attributes. Theta-join efficiency is improved here, as just part of the attributes take part in the costly operation, and input data are also filtered by MMF. They tested the methods through extensive experiments over both synthetic and real world data in a distributed cluster. Experiments show that the solution can speed up the query efficiency compared to the state-of-art solutions. The speedup ratio is ranging from 1.6 to 77.4, varying with the theta-join conditions.

Khafagy, 2015 proposed new two-way join algorithm called Indexed MapReduce Join Algorithm that used Index in the large table to decrease I/O and shuffling that cause best performance in MapReduce Join. The experimental result shows that using Index-join algorithm has high performance than other algorithms while increasing the data size from 100 million records to 500 million without memory overflow.

Blanas et al. (2010) proposed a technique for joining reference data such as information about users with log data. He described crucial implementation details of a number of well-known join strategies in MapReduce, and presented a comprehensive experimental comparison of these join techniques on a 100-node Hadoop cluster. The pre-processing techniques designed to further improve the join performance at query time. It shows that join algorithms do not require any modification to the basic Map Reduce framework and can be readily used by existing Map Reduce platforms like Hadoop.

Caching:

Hwang et al. (2018) developed resource manager named Cache-Affinity and Virtualization-Aware (CAVA), which measures the cache affinity of MapReduce applications at runtime. It efficiently manages distributed in-memory caches of a limited size by assigning high priority to applications that have high cache affinity. The proposed memory locality-aware scheduling algorithm is also integrated into the CAVA resource manager. Their extensive experimental study shows that CAVA exhibits overall good performance over various workloads composed of multiple big data analytics applications. They considered the fine-grained data locality levels in virtualized clusters and efficiently used scarce memory resources. They investigated various levels of data locality in a virtualized environment, ranging from rack locality to memory locality. Their memory locality-aware scheduling algorithm effectively increases the cache hit ratio by exploiting extra fine-grained levels of data locality in a virtualized environment. This in turn, reduces network traffic and disk I/O. However, a high cache hit ratio does not necessarily imply a shorter job execution time in MapReduce applications. To resolve this issue, they developed the Cache-Affinity and Virtualization-Aware (CAVA) resource manager.

Nanda et al. (2017) proposed CVMC (Cloud Virtual Machine Creation) algorithm, using which clients can access the cloud infrastructure through virtual machine. The clients can further install software or deploy database on cloud-based systems. The cache-size is calculated meticulously by considering the queries such as exact cache-hit queries and different percentages of cache-hit queries. It provides a technique to find out the best-suitable cache size in terms of the number of queries. The performance of cloud database with cache is most cost effective when the cache-size is kept at about 15% of the database size. The work can be further extended by including parameters like types of queries, database-size and query-set size.

Kwak et al. (2016) proposed an adaptive cache local scheduling algorithm that adaptively adjusts the waiting time of a MapReduce job in a queue for a cache local node. They set the waiting time to be proportional to the percentage of cached input data for the job. They also developed a cache affinity cache replacement algorithm that determines which block is cached and evicted based on the cache affinity of applications. Using various workloads consisting of multiple MapReduce applications, they conducted experimental study to demonstrate the effects of the proposed in-memory orchestration techniques. The experimental results show that enhanced Hadoop in-memory caching scheme improves the performance of the MapReduce workloads up to 18% and 10% against Hadoop that disables and enables HDFS in-memory

caching, respectively. An efficient scheduling algorithm can be further developed that considers the effects of co-runners and cache locality.

Sangavi et al. (2015) proposed an enhanced model for DACHE to improve the effectiveness in the MapReduce environment. The DACHE is enhanced by introducing a cache optimization technique and a better page replacement algorithm. The cache optimization technique used here is sub blocking. It reduces the miss penalty i.e. number of clocks to process a cache miss. Hence the Dache can be utilized more efficiently for MapReduce applications. The page replacement technique used is WSClock algorithm. It effectively replaces the pages by embedding a reference bit to each page. Based on the status of the bit it decides whether to evict the page or not. It maintains the oldest page which is often used by the user from being removed. It uses circular list to hold all the pages.

Mukhopadhyay et al. (2014) provided a solution to reduce the use of NameNode memory space so that the problem of NameNode memory getting full which ultimately results in hindering the scalability of cluster will be delayed. Cache is basically use for storing the frequently used data closer to the processor than the whole data and using it for faster operation can be applied in the NameNode memory management. In this separation algorithm is used which will be used for moving the least recently used metadata from NameNode to secondary storage device. In this algorithm one extra field count is added which indicated the frequency of use about each file into the cluster. Each time the access time is updated the count is increased by one indicating that the data/file represented by it is been used. In the proposed architecture, a threshold value is defined on the NameNode memory space. As and when this threshold value reaches, the separation algorithm runs so as to remove the metadata records that are not being used. Initially the NameNode stores the file system metadata as it comes, until the predefined threshold is reached. Once the threshold is reached, the separation algorithm comes into play and removes some of the records based on the separation function.

Partitioning:

Galarraga et al. (2014) designed a distributed engine (PARTOUT) for improving the performance of distribution: data partitioning, allocation, and distributed query processing, so that efficient RDF processing in a cluster of machines can be done. Partout can efficiently handle updates and its query optimizer produces efficient query execution plans for ad-hoc SPARQL queries. It proposed an effective approach for fragmenting RDF data sets based on a query log, allocating the fragments to nodes in a cluster, and finding the optimal configuration.

The experiments show the superiority of their approach to state-of-the-art approaches for partitioning and distributed SPARQL query processing. They also presented a query optimizer along with a distributed cost model and several heuristics that lead to efficient query execution plans that also exploit pipelining.

Slagter et al. (2013) proposed an improved partitioning algorithm that improves load balancing and memory consumption. They presented XTrie and ETrie, extended partitioning techniques, to improve load balancing for distributed applications. By improving load balancing, MapReduce programs can become more efficient at handling tasks by reducing the overall computation time spent processing data on each node. The TeraSort, developed by O'Malley, Yahoo, was designed based on a very large cluster of 910 nodes. In that particular computing environment and for that data configuration, each partition generated by MapReduce appeared on only one or two nodes. In contrast, their study looks at small- to medium-sized clusters. This study modifies the design and optimizes it for a smaller environment. A series of experiments showed a skewed data sample, the Etrie architecture was able to conserve more memory, can allocate more computing resources, and do so with less time complexity. Further research could be done by extending this framework so that it can use different types of micro partitioning methods for applications using different input samples.

Pipelining:

Malhotra et al. (2018) proposed a generalized model GENMR and implemented it to give solutions for processing such large sized data for cloud database management system. The model GENMR takes queries written in various RDBMS forms like SQL, ORACLE, DB2, MYSQL with which users are already familiar and with the help of model's compiler module, queries get converted into MapReduce form as MapReduce is a splendid solution to process large amount of data. Then optimized Cross Rack algorithm for placement of the Map and Reduce function is applied in such a way that it minimizes the Cross Rack Communication for getting the best results in terms of running time. In future, the proposed model could be implemented with the help of existing programming language.

Condie et al. (2010) proposed a modified MapReduce architecture that allows data to be pipelined between operators. This extends the MapReduce programming model beyond batch processing, and can reduce completion times and improve system utilization for batch jobs as well. They presented a modified version of the Hadoop MapReduce framework that supports online aggregation, which allows users to see "early returns" from a job as it is being computed.

The Hadoop Online Prototype (HOP) also supports continuous queries, which enable MapReduce programs to be written for applications such as event monitoring and stream processing. HOP retains the fault tolerance properties of Hadoop and can run unmodified user-defined MapReduce programs.

Figure 5 summaries the already existing approaches (models and algorithms) developed by various researches for processing of queries in MapReduce environment. It also indicates the directions for future research in this field in order to further improve the performance of query processing techniques in MapReduce framework.

S.No	Authors	Proposed	Technique	Experiment	Conclusion	Future Perspectives
1.	Hwang et al. 2018	Cache-Affinity and Virtualization-Aware (CAVA) resource manager, which measures the cache affinity of MapReduce applications at runtime.	Caching	Cluster consists of four physical machines. Each of which has four Intel Xeon Octa-core E5-4610v2 processors and 128 GB of memory.	CAVA resource manager improves the performance of various dynamic workloads composed of multiple MapReduce applications up to 24%.	The effects of RMML (rack machine memory locality) can be analyzed over different hardware settings for network and storage and also over different VM configurations
2.	Lui and Li. 2018	MMF (max and min value based filter) method for theta join operations. DM (Divide and Merge) for theta-join and multi-way theta-join in distributed environment.	Joining	They tested the methods over synthetic and real world data in a distributed cluster and shows that the solution can speed up the query efficiency ranging from 1.6 to 77.4, varying with the theta-join conditions.	It helps in reducing the computation cost and achieves minimum execution time in distributed environment.	Work can be done on optimization to further improve the performance of MMF and DM method.
3.	Malhotra et al. 2018	Generalized model GENMR which consists of three modules, firstly user interface, secondly the complier and lastly the enhanced optimal Cross Rack	Pipelining		It minimizes the Cross Rack Communication for achieving best results in terms of running time.	The proposed model can be implemented with the help of existing programming language in future.

		Algorithm.				
4.	Nanda et al. (2017)	CVMC algorithm, using which clients can access the cloud infrastructure through virtual machine. An appropriate cache size is calculated in terms of the number of queries.	Caching	VMC cloud server having Intel Core 2 Duo processor, 2.40 GHz CPU and 4 GB RAM, running RHEL 6 operating system.	The performance of cloud database with cache is most cost effective when the cache-size is kept at about 15% of the database size.	Work can be extended by including parameters like: types of queries and query-set size.
5.	Kwak et al. 2016	An adaptive cache local scheduling algorithm that adaptively adjusts the waiting time of a MapReduce job in a queue for a cache local node.	Caching	Cluster of 11 nodes, Ubuntu 12.04 LTS. Each node is with two Intel Xeon octa-core E5-2650, 64 GB memory and 1 TB HDD	Hadoop in-memory caching scheme improves the performance of the MapReduce workloads up to 18%	More sophisticated cache affinity metric and an efficient scheduling algorithm can be considered as the future work.
6.	Sangavi et al. 2015	An enhanced model for DACHE (Data aware cache framework) by providing a better page replacement (WS Clock algorithm) and a cache optimization technique.	Caching	The server with 8-core CPU is to run the Hadoop. Each core runs at 3GHz,16 GB memory and a SATA disk.The 16 mappers are required whereas the number of reducers varies depending on the application.	It makes searching process more easy and efficient than the existing technique and eliminating the duplicate task that co-exists with the incremental MapReduce jobs.	The complete implementation using the given setup and assessment can be considered as the future work of this research.
7.	Khafagy et al. 2015	Indexed Map Reduce Join Algorithm that used Index in the large table to Decrease I/O and Shuffling that cause best performance in MapReduce Join.	Joining	Cluster configuration consists of Intel Core I5 2.4 GHz processor, 4 GB memory for every node, 500 GB SATA disk and operating system Ubuntu 13.14 Linux with Apache hadoop release 1.2.1.	This algorithm has no limitation on the datasets size, it can join any size of data sets together at once by a foreign key. It reduces I/O and shuffling and cause enhance in total join time.	Implement join algorithms using several datasets benchmarks, increasing number of nodes to show performance and used hash semi-join idea and enhanced in this algorithm to run multi-way join.

8.	Galarraga et al. 2014	A distributed engine **(PARTOUT) f**or improving the performance of distribution: data partitioning, allocation, and distributed query processing	Partitioning	All experiments were implemented on top of RDF-3X version 0.3.612 using a cluster of machines each with an Intel Xeon E5430 processor and 32 GB of RAM running Debian GNU/Linux 6.0 for 64 bits.	The proposed techniques are suitable for the distributed setup and especially speed up queries involving large intermediate results that in combination with a beneficial allocation scheme do not have to be transferred to another host.	Combine the advantages of query-load aware partitioning and replication in one system.
9.	Mukhopadhyay et al. 2014	Proposed an enhanced architecture of Hadoop for handling the issue of NameNode scalability through cache concept.	Caching	Hadoop is deployed on three machines having Ubuntu 13.04 and creating multimode cluster. One dedicated machine for NameNode and two for DataNodes. The data set used is a collection of 1.8 million text files collected from the weather survey.	Reduce the NameNode memory consumption and increased the capacity of a Hadoop cluster.	For future read operations it can be considered as a positive perspective, since whenever read request is made search operation carried on the NameNode will return the recent data.
10.	Slagter et al. 2013	Proposed XTrie and ETrie, partitioning techniques for improving the load balancing for distributed applications.	Partitioning	Experiments Conducted were done using100eBooks, with each eBook containing a minimum of 100,000 words. Using the eBooks as input, they simulated computer networks from 5 nodes in size to 100 nodes in size.	Improved computation time for MapReduce when these methods are executed on small or medium sized cluster of computers	Further research can be done extending this framework so that it can use different micro partitioning methods for applications using different input samples.

11.	Condie et al. 2010	Developed Hadoop Online Prototype (HOP) which supports continuous queries, and enables MapReduce programs to be written for applications such as event monitoring and stream processing.	Pipelining	Demonstrate online aggregation in HOP using a data set containing seven months of hourly page view statistics for more than 2.5 million Wikipedia articles.	HOP provides new functionality, which includes "early returns" on long-running jobs via online aggregation, and continuous queries over streaming data.	As a more long-term agenda, we can explore using MapReduce style programming for even more interactive applications.
12.	Blanas et al. 2010	Proposed a technique for joining reference data such as information about users with log data.	Joining technique.	100- node Hadoop cluster. Each node had a single 2.4GHz Intel Core 2 Duo processor with 4GB of DRAM and two SATA disks.	They design pre-processing techniques to improve the join performance at query time.	An optimization module can be design in future work that can automatically select the appropriate join algorithms.

Figure 5: Review of Literature

3. Research Motivation

In today's IT world, cloud computing has become a key factor for the technological transformation towards achieving the business goals through its extensive list of advantages. While demonstrating several advantages, cloud computing also has its potential risks and challenges on the other side. Due to the uprising of new technologies like Social media, Smart phone, Internet of Things (IoT), etc. more and more data get accumulated over internet year after year which must be handled in a proper way. Around 2.5 quintillion bytes of data have been created every day in current scenario over the last two years alone 90 percent of the data in the world was generated (Bernard, 2018). The constant challenge is how to access to this information or data in an efficient manner. MapReduce paradigm has emerged as a universal tool for storing and quick processing of huge amounts of data which could be unstructured data like text, images or videos.

The research that is being done in the area of query processing largely focuses on handling of intermediate data produced by Map and Reduce node i.e. caching (Hwang et al. (2018), Kwak et al. (2016), Sangavi et al. (2015) and Mukhopadhyay et al. (2014)), managing multiple inputs through MapReduce i.e. joining (Liu et al. (2018) and Ezzaki et al. (2018), managing load

balancing over nodes (Slagter et al. (2013) and managing pipelining through MapReduce (Malhotra et al. (2018) and Condie, 2010).

Joins are not directly supported by MapReduce. Hence the main challenge of query processing based on MapReduce is still how to process join operation? For large scale data, it is still a issue that how to optimize the cache in such a way that a process should perform minimum buffer operations by making the use of cache memory? The pipelining challenge requires that producers and consumers should co-schedule intelligently in a MapReduce environment. Partitioning of the data is also a major issue since it must be done in such way that allows queries to be executed over a minimum number of nodes. There are certain limitations with the works reported in the past and the same have a scope for further improvement. These studies have acted as a motivation to develop an efficient technique of query processing using MapReduce.

4. Objectives of the Present Research

The objectives of the present research are:

1. To study the architecture of MapReduce-based systems in cloud environment.
2. To examine the factors affecting the query performance in MapReduce-based systems.
3. To review the existing techniques for processing of queries in MapReduce-based systems and select a technique for the study after identifying research gaps.
4. To propose a query processing technique for enhanced efficiency.
5. To decide the parameters for the evaluation of the proposed technique.
6. To evaluate the efficiency of the proposed technique.

5. Methodology

To achieve the above objectives, we propose to use the following methodology:

1. Study the architecture of Map Reduce-based systems in cloud environment. To perform this, we will study all the articles and latest research papers published in reputed journals.
2. Examine the factors which affect the query performance in Map Reduce-based systems. For this, we will study all the factors affecting the query performance in Map Reduce-based systems directly or indirectly.
3. To review the existing techniques for processing of queries in Map Reduce-based systems and select a technique for the study by keeping track of the latest research.
4. To enhance the efficiency of a query processing technique.

- This will be achieved either by making refinements in the existing technique or designing a new technique.
- For this, we will identify the gaps of previous work performed in the field of query processing through MapReduce-based systems.
- Select an existing algorithm and find the area of improvement if possible or suggest the designing of new algorithm for fast and accurate retrieval of information.

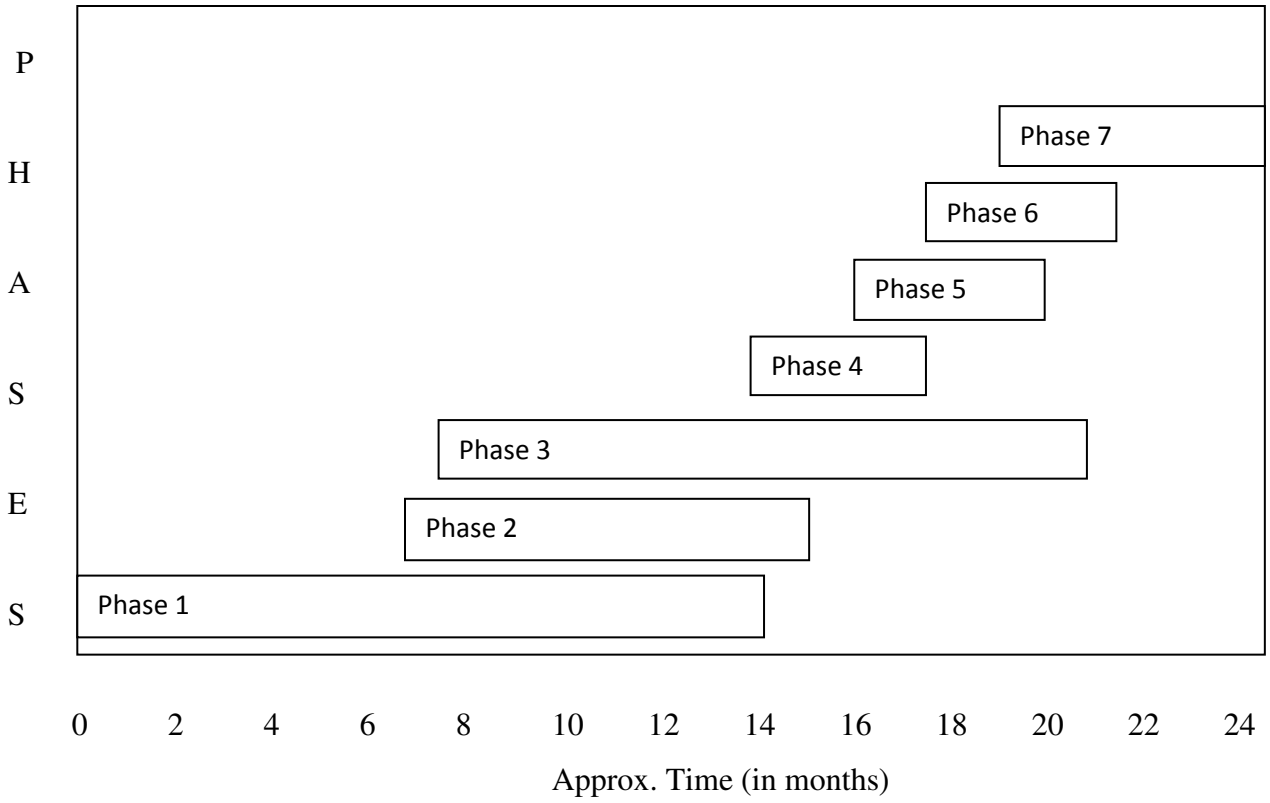
5. Decide the parameters for evaluation of proposed technique. For this, we have to identify all the parameters and select a parameter which will further improve the efficiency of proposed algorithm.
6. Comparison and evaluation of the performance of proposed technique for establishing its validity.
7. Presentation of results in the form of research papers and thesis reports.

Plan of Work

The work plan indicates the projected time period required in the research. We have divided the research work into different phases which are as follows:

Phase No.	**Phase Description**	**Approximate time duration (in months)**
1	Study the architecture of MapReduce-based systems in cloud environment.	8-14
2	Examine the factors affecting the query performance in MapReduce-based systems	2-8
3	Review the existing techniques for processing of queries in MapReduce-based systems and select a technique for the study after identifying research gaps	5-15
4	Propose a query processing technique for enhanced efficiency.	3-4
5	Decide the parameters for the evaluation of proposed technique.	4-5
6	Evaluate the efficiency of the proposed technique.	4-5
7	Report generation	5-6

The timeline chart for the above phased plan is as follows:



6. Tools and Techniques

The tools and technologies that may be used to carry out the proposed research include:

- Open Source Databases: HBase, CouchDB, LucidDB, etc.
- Datasets: Repositories (Data.gov, UCI).
- Graphical Tools: Gliffy, Draw Anywhere, etc.
- Statistical Tools: MS-Excel, SPSS.
- Simulator: CloudSim, CloudAnalyst, etc
- TPC Standards: TPC-E, TPCx-BB, TPCx-HS, etc.

7. References

- Akhtar, N., F. Parwej, and Y. Perwej, (2017). A Perusal of Big Data Classification and Hadoop Technology. *International Transaction of Electrical and Computer Engineers System, Science and Education*, **4**(1), 26-38.
- Almorsy, M., J. Grundy, and I. Müller, (2016). An analysis of the cloud computing security problem. *In Proceeding of the Asia Pacific Cloud Workshop,* Sydney, Australia, May 30, 2016.
- Bachhav, A., V. Kharat, and M. Shelar, (2017). Query Optimization for Databases in Cloud Environment: A Survey. *International Journal of Database Theory and Application* **10**(6), 1-12.

- Blanas, S., J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian, (2010). A comparison of join algorithms for log processing in MapReduce. *In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, ACM,* held at Indiana, USA from June 6 to June11, 2010, 975-986.

- Bobrowski, S. (2011), The Future of Databases is in the Clouds. http://wiki.database.com/page/The Future of Databases is in the Clouds.

- Condie, T., N. Conway, P. Alvaro, J. M. Hellerstein, J. Gerth, J. Talbot, K. Elmeleegy, and R. Sears, (2010). Online aggregation and continuous query support in MapReduce. *In SIGMOD Conference,* 1115–1118.

- Doulkeridis, C., and K. NØrvåg, (2014). A survey of large-scale analytical query processing in MapReduce. *The VLDB Journal- The International Journal on Very Large Data Bases,* ***23***(3), 355-380.

- Elmasri, R. and S. Navathe, (2016). *Fundamentals of database systems. London: Pearson*.

- Galárraga, L., K. Hose, and R. Schenkel, (2014). Partout: a distributed engine for efficient RDF processing. *In Proceedings of the 23rd International Conference on World Wide Web ACM,* held at Seoul, Korea from April 7 to April 11, 2014, 267-268.

- https://www.forbes.com/sites/bernardmarr/2018/05/21.

- Hwang, E., H. Kim, B. Nam and Y. Choi, (*2018).* CAVA: Exploring Memory Locality for Big Data Analytics in Virtualized Clusters, *18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, held at Washington, DC, USA from at May 1 to May 4, 2018, 21-30.

- Khafagy, M. H. (2015). Indexed map-reduce join algorithm. *International Journal of Scientific & Engineering Research*, ***6***(5), 705-711.

- Kwak, J., E. Hwang, T. K. Yoo, B. Nam, and Y. R. Choi, (2016). In-Memory Caching Orchestration for Hadoop. InCluster, Cloud and Grid Computing (CCGrid), 16th *IEEE/ACM International Symposium,* held at Cartagena, Colombia from May 16 to May 19, 2016, 94-97.

- Liu, W., and Z. Li, (2018). An efficient theta-join query processing in distributed environment. *Journal of Parallel and Distributed Computing,* **121**, 42–52.

- Malhotra, S., M. N. Doja, B. Alam, and M. Alam, (2018). Generalized Query Processing Mechanism in Cloud Database Management System. *In Big Data Analytics, Advances in Intelligent Systems and Computing, Springer, Singapore,* ***654****,* 641-648.

- Mukhopadhyay, D., C. Agrawal, D. Maru, P. Yedale, and P. Gadekar, (2014). Addressing Name Node Scalability Issue in Hadoop Distributed File System Using Cache Approach. *In Information Technology (ICIT), International Conference IEEE,* 321-326.

- Nanda, R., (2015). Review of Query Processing Techniques of Cloud Databases. *Suresh Gyan Vihar University Journal of Engineering & Technology,* **1**(2), 12-16

- Nanda, R., K.S. Sharma and S. Chande, (2017). Determining Appropriate Cache-size for Cost-effective Cloud Database Queries. *International Journal of Computer Applications,* **157**(6), 29-34.

- Mell, P. and T. Grance, (2015)" The NIST definition of cloud computing" (draft), National Institute of Standards and technology (NIST), Gaitherburg, MD.
- Prasad, G. (2017). An Efficient Approach to Optimize the Performance of Massive Small Files in Hadoop MapReduce Framework. *JCSE International Journal of Computer Sciences and Engineering, E-ISSN:2347-2393,***5**(6), 112-120
- Sangavi, S., A. Vanmathi, R. Gayathri, R. Raju, P. V. Paul, and P. Dhavachelvan, (2015). An Enhanced DACHE Model for the MapReduce Environment. *Procedia Computer Science, **50***, 579-584.
- Silberschatz, A., Henry, F. Korth, S. Sudarshan, (2011). *DATABASE SYSTEM CONCEPTS, Sixth Edition, Tata McGraw Hill, New Delhi.*
- Slagter, K., C. H. Hsu, Y. C. Chung, and D. Zhang, (2013). An improved partitioning mechanism for optimizing massive data analysis using MapReduce. *The Journal of Supercomputing, Springer, **66***(1), 539-555.
- White, T. (2012). *Hadoop: The Definitive Guide*, Third Edition," O'Reilly Media, Inc.", Yahoo Press, Cambridge.
- Yang, H. C., A. Dasdan, R. L. Hsiao, and D. S. Parker, (2007). Map-reduce-merge: simplified relational data processing on large clusters. *In Proceedings of the 2007 ACM SIGMOD international conference on Management of data, ACM,* held at Beijing, China from June 12 to June 14, 2007, 1029-1040.